Pi-calculus and LCC, a Space Odyssey

Sylvain Soliman

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Pi-calculus and LCC, a Space Odyssey

Sylvain Soliman

N° 4855

Juin 2003

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4855--FR+ENG

Pi-calculus and LCC, a Space Odyssey

Sylvain Soliman

Thème 2 — Génie logiciel
et calcul symbolique
Projet Contraintes

Rapport de recherche n° 4855 — Juin 2003 — 19 pages

Abstract: We present a translation of the asynchronous pi-calculus into linear concurrent constraint languages (LCC), and use that translation and the recent advances in the logical semantics of LCC to give an account of the restriction operator in Intuitionistic Linear Logic. This allows us to express as a Linear Logic theory, a notion of space similar to what has recently been introduced by Gabbay and Pitts in modal logics. It also permits to relate more closely the pi-calculus and CC paradigms that people have wanted to compare for a long time.

Key-words: Pi-calculus, concurrent constraint programming, linear logic

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Pi-calcul et LCC, une odyssée de l'espace

Résumé : Nous présentons une traduction du pi-calcul asynchrone vers les langages linéaires concurrents avec contraintes (LCC), puis utilisons cette traduction et les avancées récentes de la sémantique logique de LCC pour obtenir une représentation de l'opérateur de restriction en Logique Linéaire Intuitionniste. Ceci nous permet d'exprimer comme théorie en Logique Linéaire, une notion d'espace similaire à celle récemment introduite par Gabbay et Pitts pour la logique modale. Une autre conséquence est une meilleure compréhension des relations entre pi-calcul et langages CC, qui ont été soumis à comparaison depuis longtemps.

Mots-clés : Pi-calcul, programmation concurrente avec contraintes , logique linéaire

1 Introduction

The π-calculus has become, since its introduction [23], a standard paradigm for specifying concurrent processes. The asynchronous version introduced by Boudol [5], though it has been proved less expressive [24], allows a more realistic behavior for distributed systems and has therefore been used as basis for those, for instance in the join-calculus [12].

The tools used to analyze these languages, remain however centered around the notion of (bi-)simulation, which, being very operational, is not always satisfactory in terms of expressing declaratively certain high level properties about concurrent processes. The main purpose of this article is to provide a logical semantics to the asynchronous π-calculus , and thus the possibility to use new tools to reason about processes.

There is a long tradition of comparison between the Concurrent Constraint Programming family of languages [27] and different variants of the π-calculus : CC was from the beginning considered as a process calculus, however the fact that the constraint store was always growing monotonically made very inconvenient the encoding of the message passing primitives of the other well known process calculi [19]. The development, in the last decade, of the Linear Concurrent Constraint paradigm (LCC) that generalizes CC simply by allowing constraint systems based on Linear Logic [14] instead of classical logic, greatly extends the expressive power of CC, as non monotonic evolutions of the store are possible through the consumption of constraints by ask agents [28, 4, 11]. This generalization allows, as we shall see, for a much more natural encoding of process calculi like for instance the π-calculus , as was already suggested more than ten years ago in [28].

There have also been lots of research about the LL based logic programming models of concurrency (for instance [18]), and there is a long history relating LL and the CC family as will be briefly recalled in section 2.2; finally there is one main work (though it was labelled "preliminary results") relating directly the reduction in π-calculus and theories of LL: that of Miller in [21].

After some preliminary section for introducing the notations used throughout the article, the paper is thus articulated around two encodings.

First, we give in section 3 a simple, compositional, sound and complete encoding of the asynchronous π-calculus into LCC (without resorting to higher order). The impossibility of a reverse embedding is also discussed. This first step already gives rise to a new way of reasoning about π-calculus processes, namely translating them into LCC and using there the tools developed for that framework [10, 11, 29].

We then build on this natural encoding to take into account the latest version of the logical semantics of LCC [30]: one of the interesting points about this semantics, based on the Intuitionistic Linear Logic, is that it manages to capture the restriction operator of LCC through a simple theory of ILL. This operator, which has always been a problem from the logics side [8], was formerly identified with the existential quantifier of the underlying logic, with not completely satisfactory results (detailed in section 4.1). It has recently been proposed to add new operators to capture more precisely the meaning of the restriction operator, the most advanced proposal using *binders* [13], and allowing a precise capture of the spatial notion of restriction in process calculi [6]. We propose here to use the same

construction used for the logical semantics of LCC to give a characterization of the spatial nature of the π-calculus by a theory of ILL, thanks to the encoding mentioned above. This last point is to be related with the work of Miller [21] which already succeeded in representing the π-calculus as a theory in LL, however we manage to go one step further than that work as we encode *even the restriction operator* in the theory, in contrast with using sequents equipped with a signature.

This Linear Logic theory of the asynchronous π-calculus gives us, in turn, new tools for reasoning about the processes. Section 5 starts to describe how these tools can be put to work; the article is then concluded by a discussion about the strong links between π-calculus and LCC, space and linear logic, and a few words concerning the future work and perspectives in these domains.

2 Preliminaries

2.1 The asynchronous π-calculus

We briefly recall the syntax and semantics of the asynchronous π-calculus , as given in [5]:

Let $\mathcal{N}$ be an infinite set of *names*, ranged over by $x, y, z, \ldots$, the context-free syntax of *processes* is the following:

$$\begin{array}{rl} P ::= & \bar{x}z \\ & x(y)P \\ & P \mid P \\ & !P \\ & (\nu x)P \end{array}$$

The intuitive meaning of these constructs being that $\bar{x}z$ outputs the message z on the channel x, $x(y)P$ receives a message on channel x and then executes the process P where y represents the received message, $P \mid P$ runs the two processes in parallel, $!P$ replicates the process P and $(\nu x)P$ restricts the scope of the name x in P.

We shall see in the next sections that the last two of these constructors, namely ! and ν are the ones that are the most difficult to characterize logically. However the restriction operator ν is crucial to the expressivity of the π-calculus , especially as the sharing of names can be seen as a structuring of the processes in *spatial* regions, the mobility of names allowing the jump from CCS to π-calculus and thus for instance an easy encoding of the λ-calculus; it should therefore not be neglected.

The input and restriction constructors $x(y)P$ and $(\nu y)P$ both bind the name y in P. As usual, substituting a name y for a name x in a process P, yielding $P[y/x]$, may require to rename some bound names to avoid unduly binding y. We shall denote by $fn(P)$ the set of free names of the process P, defined as usual.

The operational semantics is given in the style of the CHAM [3], with $\equiv$, the *structural congruence*, defined as the smallest congruence on processes, satisfying the rules of table 1.

(1a)	$P \mid Q \equiv Q \mid P$	
(1b)	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	
(2a)	$(\nu x)P \mid Q \equiv (\nu x)(P \mid Q)$	$(x \notin fn(Q))$
(2b)	$(\nu x)P \equiv (\nu y)P[y/x]$	$(y \notin fn(P))$
(2c)	$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$	
(2d)	$(\nu x)P \equiv P$	$(x \notin fn(P))$
(3)	$!P \equiv P \mid !P$	

Table 1: Structural Congruence

The transition relation $\longrightarrow$ is the least binary relation satisfying the rules of table 2.

(1)		$x(y)P \mid \bar{x}z \longrightarrow P[z/y]$
(2)	$P \longrightarrow P' \Rightarrow$	$P \mid Q \longrightarrow P' \mid Q$
(3)	$P \longrightarrow P' \Rightarrow$	$(\nu x)P \longrightarrow (\nu x)P'$
(4)	$P \equiv P', P \longrightarrow Q, Q \equiv Q' \Rightarrow$	$P' \longrightarrow Q'$

Table 2: Transition Relation

2.2 Preliminaries on LCC languages

There are several reasons to consider constraint systems based on linear logic instead of classical logic in the study of CC languages. First, as shown in [28, 10, 11] linear logic provides a faithful logical semantics to CC agents, it is therefore natural to adopt the same logic for agents and for constraints. Second, thanks to the standard translation of classical logic into linear logic, linear constraint systems are a generalization of classical constraint systems, that is any classical constraint system can be presented as a linear constraint system. Third, in order to escape from the fact that constraints can only be added to the store along a computation in CC languages, several variants of CC where the constraints can be consumed by ask agents and thus removed from the store, have been introduced by Saraswat and Lincoln [28, 31] or Best, de Boer and Palamidessi [4]: these variants enhance significantly the expressive power of CC and the constraints are naturally modeled as formulae of linear logic. In this section we present the variant LCC used in [10, 11].

2.2.1 Syntax

In this paper, a set of variables is denoted by X, Y,..., the set of free variables occurring in a formula A is denoted by $fv(A)$, a sequence of variables is denoted by $\vec{x}$, $A[\vec{t}/\vec{x}]$ denotes the formula A in which the free occurrences of variables $\vec{x}$ have been replaced by terms $\vec{t}$ (with the usual renaming of bound variables, avoiding variable clashes).

For a set S, $S^\star$ denotes the set of finite sequences of elements in S. For a transition relation $\Rightarrow$, $\Rightarrow^\star$ denotes the transitive and reflexive closure of $\Rightarrow$.

The essential difference between LCC and CC is that constraints are formulae of linear logic and that communication (the *ask* rule) consumes information.

Definition 2.1 (Linear constraint system) *A* linear constraint system *is a pair* $(\mathcal{C}, \vdash_\mathcal{C})$, *where:*

- $\mathcal{C}$ *is a set of formulae (the* linear constraints*) built from a set* V *of variables, a set* Σ *of function and relation symbols, with logical operators: the multiplicative conjunction* $\otimes$*, its neutral element* 1*, and the existential quantifier* $\exists$*;*
- $\Vdash_\mathcal{C}$ *is a subset of* $\mathcal{C} \times \mathcal{C}$ *which defines the non-logical axioms of the constraint system.*
- $\vdash_\mathcal{C}$ *is the least subset of* $\mathcal{C}^\star \times \mathcal{C}$ *containing* $\Vdash_\mathcal{C}$ *and closed by the following rules of intuitionistic linear logic (see appendix A for the complete sequent calculus):*

$$c \vdash c \qquad \frac{\Gamma, c \vdash d \quad \Delta \vdash c}{\Gamma, \Delta \vdash d} \qquad \vdash 1 \qquad \frac{\Gamma \vdash c}{\Gamma, 1 \vdash c}$$

$$\frac{\Gamma \vdash c_1 \quad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \qquad \frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c}$$

$$\frac{\Gamma \vdash c[t/x]}{\Gamma \vdash \exists x\ c} \qquad \frac{\Gamma, c \vdash d}{\Gamma, \exists x\ c \vdash d}\ x \notin fv(\Gamma, d)$$

The syntax of LCC *agents* is given in table 3, where $\|$ stands for parallel composition, $+$ for non-deterministic choice, $\exists$ for variable hiding and $\rightarrow$ for blocking ask. The atomic agents $p(\vec{x})\ldots$ are called *process calls* or *procedure calls*, we assume that the arguments in the sequence $\vec{x}$ are all distinct variables. The *ask* agent in LCC is written with a universal quantifier in order to make explicit the variables which are bound in the guard.

Recursion is obtained by declarations. We make the usual hypothesis that in a declaration $p(\vec{x}) = A$, all the free variables occurring in A occur in $\vec{x}$. The set of declarations of an LCC program, denoted by D, is the closure by variable renaming of a set of declarations given for distinct procedure names p. A *program* $\mathcal{D}.A$ is a set of declarations $\mathcal{D}$ together with an initial agent A.

2.2.2 Operational semantics

The operational semantics is defined on configurations where the store is distinguished from agents. A *configuration* is a triple $(X; c; A)$, where c is a constraint called the store, A is an agent or $\emptyset$ if empty, and X is a set of variables, called the *hidden* variables of c and A. The operational semantics is defined in the style of the CHAM [3] by a transition system which does not take into account specific evaluation strategies. The *structural congruence* $\equiv$ is the least congruence satisfying the rules of table 3. For convenience here, and unlike in [11], the logical equivalence of constraints is not built-in in the congruence. We write $\Gamma, \Delta, \ldots$ for

Agents	$A ::= p(\vec{x}) \mid tell(c) \mid (A \parallel A) \mid A + A \mid \exists x A \mid \forall \vec{x}(c \rightarrow A)$
Declarations	$\mathcal{D} ::= \epsilon \mid p(\vec{x}) = A \mid \mathcal{D}, \mathcal{D}$
Program	$P ::= \mathcal{D}.A$
α-Conversion	$\dfrac{z \notin fv(A)}{\exists y A \equiv \exists z A[z/y]}$
Parallel comp.	$A \parallel B \equiv A, B$
Equivalence	$\dfrac{(X;c;\Gamma) \equiv (X';c';\Gamma') \longrightarrow (Y';d';\Delta') \equiv (Y;d;\Delta)}{(X;c;\Gamma) \longrightarrow (Y;d;\Delta)}$
Tell	$(X;c;tell(d),\Gamma) \longrightarrow (X;c \otimes d;\Gamma)$
Ask	$\dfrac{c \vdash_{\mathcal{C}} d[\vec{t}/\vec{y}] \otimes e}{(X;c;\forall \vec{y}(d \rightarrow A),\Gamma) \longrightarrow (X;e;A[\vec{t}/\vec{y}],\Gamma)}$
Hiding	$\dfrac{y \notin X \cup fv(c,\Gamma)}{(X;c;\exists y A,\Gamma) \longrightarrow (X \cup \{y\};c;A,\Gamma)}$
Procedure calls	$\dfrac{(p(\vec{y}) = A) \in D}{(X;c;p(\vec{y}),\Gamma) \longrightarrow (X;c;A,\Gamma)}$
Local choice	$(X;c;A+B,\Gamma) \longrightarrow (X;c;A,\Gamma)$ $(X;c;A+B,\Gamma) \longrightarrow (X;c;B,\Gamma)$

Table 3: LCC syntax and operational semantics.

multisets of agents in configurations. Congruence is extended to multisets of agents in the obvious way: $\Gamma \equiv \Gamma'$ iff $\Gamma = \{A_1, \ldots, A_n\}$, $\Gamma' = \{A'_1, \ldots, A'_n\}$ and $\forall i = 1, \ldots, n$, $A_i \equiv A'_i$. Two configurations are said *congruent*, $(X;c;\Gamma) \equiv (X';c';\Gamma')$, when the sets X and X' are equal, the constraints c and c' are $\mathcal{C}$-equivalent, and the multisets of agents Γ and Γ' are congruent. The *transition* relation $\longrightarrow$ is the least transitive relation on configurations satisfying the rules of table 3.

We will call *store* of a configuration $(X;c;\Gamma)$ the constraint $\exists X c$.

3 Encoding π into LCC

In this section we give an encoding of the asynchronous π-calculus into LCC, following the ideas of [28], however we do not need here to resort to higher order. Indeed there is a one to one correspondence between π-calculus and LCC operators that lead to a very simple and natural translation.

One should note that the usual choice is to use a *local* operator for the non-determinism in LCC, this could allow an encoding of a corresponding operator for the π-calculus , however, even the *guarded* version of the choice operator of LCC, is only *input-guarded* and is thus not able to replicate the *global* choice of the (synchronous) π-calculus , this is why only the standard (deterministic) asynchronous π-calculus will be treated here[1]. One might also want to use a non-standard version of LCC, with synchronous *tell* and *ask*, however that kind of decision, besides being unnatural (see most papers about non-monotonic versions of CC [28, 31, 4, 11]) would lead to a failure similar to that encountered in [19] when comparing CC with the π-calculus .

The encoding of the messages is the one suggested in [28], obtained by adding all messages to the store and retrieving them by matching the channel name.

The translation is given in table 4, for the Herbrand constraint system and with declarations $\mathcal{D}$.

$$
\begin{array}{rcl}
[\bar{x}z] & = & tell(msg(x,z)) \\
[x(y)P] & = & \forall y(msg(x,y) \rightarrow [P]) \\
[P \mid Q] & = & [P] \parallel [Q] \\
[!P] & = & bangP \\
 & & \text{with } \; bangP = [P] \parallel bangP \text{ in } \mathcal{D} \\
[(\nu x)P] & = & \exists x([P])
\end{array}
$$

Table 4: Translation of the asynchronous π-calculus into LCC

This compositional translation is both sound and complete with respect to the two transition systems:

Proposition 3.1 (Soundness) *Let P and Q be asynchronous π-calculus processes, if $P \longrightarrow Q$ then there exist X, Γ and Q' such that: $(\emptyset; 1; [P]) \longrightarrow^* (X; 1; \Gamma)$,*

with $Q' \equiv Q$ using only the rule $!R \equiv !R \mid R$, and $(\emptyset; 1; [Q']) \longrightarrow^* (X; 1; \Gamma)$ using only hiding and equivalence rules.

Proof.

We reason by case on the reduction $P \longrightarrow Q$. Rules 1 and 2 of table 2 have immediate counterparts in LCC, noting that some of the possible **Tell** rules may have to be applied before an **Ask**, even if those reductions correspond to nothing in π-calculus .

[1] See also section 4.1 for a reason why local choice is not very interesting in the asynchronous π-calculus

For rule 3, the **Hiding** rule needs to be applied, which is the reason we do net get exactly $(\emptyset; 1; [Q'])$.

For rule 4, we can simply remark that most rules of table 2 correspond to the definition of $\equiv$ in LCC, except rule 3, handled by the **Procedure calls** rule, when it is in the right direction, or leading to the difference between Q and Q' otherwise. □

Proposition 3.2 (Completeness) *Let P and Q be asynchronous π-calculus processes, if $(\emptyset; 1; [P]) \longrightarrow^* (X; 1; \Gamma)$, and $(\emptyset; 1; [Q]) \longrightarrow^* (X; 1; \Gamma)$ using only hiding rules, then $P \longrightarrow^* Q$.*

Proof.

Reasoning by induction on the length of the LCC reduction, and then by case on the last step, we get an even more direct correspondence once we notice that as we have taken Herbrand constraint system, it is now straightforward to check that only a message sent on channel x (i.e. $msg(x, z)$) can entail the guard $\forall y(msg(x, y) \rightarrow A)$ and thus unblock the ask; that if a **tell** rule is fired without the corresponding **ask**, we do not get $(X; 1; \Gamma)$ (the store cannot be empty); that **procedure calls** will be mimicked by replications; and thus that the only rules amounting to a difference will be **hiding** rules. □

It is worth noticing that there have been attempts at a reverse encoding, more specifically, at embedding CC into some variants of the π-calculus , see for instance [32] for a fusion-calculus encoding of name equations and inequations. However such a reverse encoding, can only capture a small part of the CC paradigm by restricting the constraint domain (basically to Herbrand), which is the main difference between CC and some languages like Linda.

Moreover, the ask operation of LCC does allow some very powerful consumption of resources, as it permits an *atomic* consumption of two (or more) constraints, leading for instance to trivial solutions of problems such as the *dining philosophers* as it is possible to grab two forks atomically. These two points are the main arguments forbidding such a reverse encoding.

4 A theory for space in Linear Logic

The recent results concerning the logical semantics of LCC (cf. [30] for details and proofs) suggest that the existential quantifier of LL is sufficient to properly encode the restriction operator of LCC, and thus thanks to the above encoding, that of the π-calculus .

Along the lines of what is done in [21] for the synchronous π-calculus , we will thus define a translation of the asynchronous π-calculus processes into LL formulae. Our goal being to identify the reflexive and transitive closure of $\longrightarrow$ with the entailment relation $\vdash$.

4.1 Technical issues of the encoding

The idea of encoding ν by $\exists$ is not new, however it usually results in sound but not complete encodings. The problem lies in the right rule for $\exists$ (but both left and right rules are necessary

for the soundness as shown later in lemma 4.1):

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

If we want to interpret this as π-calculus processes, we get that if $P \longrightarrow^* Q[t/x]$ then $P \longrightarrow^* (\nu x)Q$, which is obviously false, for instance with $\bar{x}z \not\longrightarrow^* (\nu y)\bar{x}y$.

However, we do not want to change the logics by adding a second part to the sequents to hold a *signature* as was done in [21] (to take into account hidden variables). Using some new connectives would also be a solution to characterize the restriction operator; a good candidate would then be the *freshness* operator of [13, 25], as used in [6], but as we shall see the usual operators of IMLL are sufficient and allow us to be able to keep using the tools of LL to reason about the resulting semantics (see for instance [29] for some use of the phase semantics of LL to check the correctness of protocols expressed in LCC) without falling into the usual trap of the $\exists$ encoding explained above.

We cannot use here a translation of $x(y)P$ using $\forall$ and $\multimap$ because of completeness reasons similar to those explained for the semantics of LCC in [11]. Basically the right implication rule leads to *false receivers*, for instance we have $A, msg(x,y) \vdash A \otimes msg(x,y)$ and thus $A \vdash msg(x,y) \multimap msg(x,y) \otimes A$ which of course does not correspond to anything in π-calculus . For this reason we stick with a second order based translation even if this problem could probably be avoided with non-commutative logic as in [30].

The second order is also necessary for ! as it is well known that the ! of LL cannot replicate that of π-calculus .

Finally, it would be possible in LL also to encode a *local* version of the choice operator, which is what was done in [21], but we do not feel that it would be worth explaining in more details, and will therefore stick with the asynchronous π-calculus ; especially since the *local* choice can be embedded in the (deterministic) asynchronous π-calculus like that:

$$P + Q ::= (\nu x)(\nu y)\ (\bar{x}y \mid x(z).P \mid x(z).Q)$$

4.2 The theory

The processes of the asynchronous π-calculus are then translated following the inductive definition of table 5 into formulae built with the three atoms rcv, msg and loc of respective arity 3 (third argument is second order), 2 and 1.

$$\begin{array}{rcl} [\![\bar{x}z]\!] & = & msg(x,z) \\ [\![x(y)P]\!] & = & rcv(x,y,[\![P]\!]) \\ [\![P \mid Q]\!] & = & [\![P]\!] \otimes [\![Q]\!] \\ [\![!P]\!] & = & ![\![P]\!] \\ [\![(\nu x)P]\!] & = & \exists x(loc(x) \otimes [\![P]\!]) \end{array}$$

Table 5: Linear Logic Semantics

Where rcv and ! are defined using second order as in [21] (we use the same ! notation here to be consistent, however it should not be confused with the "of course" operator of LL noted in the same way in appendix A):

$rcv(x, y, P) \otimes msg(x, z) \vdash P[z/y]$

$!P \dashv\vdash P \otimes !P$

We also impose the following non-logical axiom:

$\exists x(loc(x)) \dashv\vdash 1$.

These are all the axioms forming our simple LL theory, and with the above encoding, they are enough to encode precisely the asynchronous π-calculus as will be shown in the following sections.

4.3 Soundness

We will first focus on soundness results, which although often quite trivial, are the most important when needing to use (practically) the semantics, for instance for proving safety properties (see for instance [10] for a discussion about this issue).

Lemma 4.1 (Equivalence) *Let P and Q be some asynchronous π-calculus processes, if $P \equiv Q$ then $[\![P]\!] \dashv\vdash [\![Q]\!]$.*

Proof.

We reason by case on the rules of table 1, our translation being based on that of Miller, we will concentrate on the differences and thus detail only the rules 2a to 2d.

Rule 2a leads us to prove that $\exists x(loc(x) \otimes R) \otimes S \dashv\vdash \exists x(loc(x) \otimes R \otimes S)$ when $x \notin fn(S)$, the $\vdash$ part is obtained by a left $\exists$ rule, followed by a right $\exists$ rule (with t chosen to be x) and axioms. The $\dashv$ part follows exactly the same proof structure.

Rule 2b leads us to prove that $\exists x(loc(x) \otimes R) \dashv\vdash \exists y(loc(y) \otimes R[y/x])$ when $y \notin fn(R)$. The proof structure is exactly the same as above, but t is now chosen to be x for $\vdash$ and y for $\dashv$.

Rule 2c is done in a similar manner, using the same technique as 3c to get x and y different, and then the same technique as 3a to get the $loc(x)$ inside $\exists y$ and reciprocally.

Rule 2d, is a direct consequence of left $\exists$ rule and of the non-logical axiom $\exists x(loc(x)) \dashv\vdash 1$.

□

Theorem 4.2 (Soundness) *Let P and Q be some asynchronous π-calculus processes, if $P \longrightarrow^* Q$ then $[\![P]\!] \vdash [\![Q]\!]$.*

Proof.

By induction on the transition $P \longrightarrow^* Q$.

The base case $P \longrightarrow^* P$ is obvious (axiom rule).

For the induction, we proceed by case on the last transition $P \longrightarrow^* R \longrightarrow Q$, noting that by induction hypothesis we already have $[\![P]\!] \vdash [\![R]\!]$ and that using the cut rule we only need to prove that $R \longrightarrow Q$ implies $[\![R]\!] \vdash [\![Q]\!]$.

We prove that by induction on the size of the proof (for $\Rightarrow$) that $R \longrightarrow Q$; the basic case corresponds to rule 1 of table 2, and the induction is obtained by case on $R \longrightarrow Q$ with rules 2 to 4. We detail here the rules 1 and 4, the other ones being trivial applications of the induction hypothesis.

The case of rule 1 is easily obtained from the application of the non-logical axiom for rcv.

The case of rule 4 is the following: we need to prove that $[\![R]\!] \vdash [\![Q]\!]$ knowing that $R \equiv R', R' \longrightarrow Q'$ and $Q' \equiv Q$. From lemma 4.1 we get $[\![R]\!] \dashv\vdash [\![R']\!]$ and the same for Q, and by induction hypothesis we have $[\![R']\!] \vdash [\![Q']\!]$, we then just need to apply twice the cut rule to get the expected result.

□

We will now tackle the more complex completeness results, giving us a perfect match between the transition relation and its logical counterpart.

4.4 Completeness

We will start by some preliminary remarks on the structure of the proofs obtained by translation of π-calculus processes.

Lemma 4.3 (Rule grouping) *If $[\![P]\!] \vdash [\![Q]\!]$ is provable, then there is a proof of that sequent with all left $\exists$ rules grouped together with the corresponding left $\otimes$ rule, and the same goes for the right rules.*

Proof.

Let us start with the case of right $\exists$:

$$\frac{\Gamma, \Gamma' \vdash loc(x) \otimes \Delta}{\Gamma, \Gamma' \vdash \exists x(loc(x) \otimes \Delta)}$$

All the rules above that one that are not right $\otimes$ rules only act on the left part of the sequent; as there is no condition on left rules, they could have been applied before the right $\exists$, there is thus another proof of the same sequent with the right $\otimes$ just above the right $\exists$ rule.

For the left rule:

$$\frac{\Gamma, loc(x) \otimes \Delta \vdash \Gamma'}{\Gamma, \exists x(loc(x) \otimes \Delta) \vdash \gamma'}$$

The same kind of reasoning applies: rules on the right can be swapped without any trouble; on the left more rules can be applied on the lower sequent than on the upper one (less free variables), and thus rules that do not touch the $\otimes$ can be swapped freely too.

All this proof can also be seen as a simple consequence of a generalized version of focusing proofs [1] ($\exists$ and $\otimes$ have the same *synchrony*), for IMLL with some non-logical axioms. □

Theorem 4.4 (Completeness) *Let P and Q be some asynchronous π-calculus processes, if $[\![P]\!] \vdash [\![Q]\!]$ then $P \longrightarrow^* Q$.*

Proof.

We reason by induction on the proof of $[\![P]\!] \vdash [\![Q]\!]$. More precisely, we can notice that in the proof, the right side of the sequent will always be of the form $[\![R]\!]$ (using lemma 4.3 for the right $\exists$ rule), and that on the left we can always replace commas by $\otimes$ and put an $\exists x$ around if there is a $loc(x)$ to get the translation of a process, the induction thus makes sense. Note that we have to allow cuts on non-logical axioms in the proofs, but that this does not invalidate the above reasoning.

For the base case, if the last rule is an axiom then we get the result by reflexivity of $\longrightarrow^*$.

The case of the non-logical axioms is also immediate as they correspond directly with their π-calculus counterpart.

The case of the cut rule is directly derived from the transitivity of $\longrightarrow^*$.

If the proof ends with a right $\otimes$ rule (thanks to lemma 4.3, the case of the $\otimes$ inside an $\exists$ can be treated separately), we have $[\![P]\!] = [\![P_1]\!] \otimes [\![P_2]\!]$ thus $P = P_1 \mid P_2$ and the same for Q with Q_1 and Q_2. By induction hypothesis we have $P_1 \longrightarrow^* Q_1$ and $P_2 \longrightarrow^* Q_2$ thus by applying repeatedly rules 2 of table 2 and 1a of table 1 we get $P_1 \mid P_2 \longrightarrow^* Q_1 \mid Q_2$, qed.

The case of the left $\otimes$ is immediate.

If the proof ends with a right $\exists$ rule, preceded immediately by a right $\otimes$ rule, thanks to lemma 4.3, we get:

$$\frac{\Gamma \vdash loc(x) \quad \Delta \vdash [\![Q_1]\!]}{\Gamma, \Delta \vdash \exists x(loc(x) \otimes [\![Q_1]\!])}$$

We get that Γ necessarily equals $loc(x)$ as there is no logical axiom concerning $loc(x)$ and no way to add an $\exists$ in front of it. We can now use the induction hypothesis on Δ and Q_1 and conclude by noticing that from applying repeatedly rule 4 of table 2 if $R \longrightarrow^* R'$ then $(\nu x)R \longrightarrow^* (\nu x)R'$, qed.

The case of the left $\exists$ is immediate.

□

5 Using a Linear Logic theory for π

Providing a theory for the asynchronous π-calculus in Linear Logic allows us to use all the tools of LL to reason about π-calculus processes. In this section we will give examples of how that can help understanding better the behavior of some π-calculus agents.

5.1 Proof search

The first basic use of Logics to reason about programs is proof search. In our context, one can note that the completeness theorem 4.4 can allow us to reason about liveness properties of π-calculus processes ($[\![P]\!] \vdash [\![Q]\!]$ implies $P \longrightarrow^* Q$), and the soundness theorem 4.2 about safety properties ($[\![P]\!] \nvdash [\![Q]\!]$ implies $P \not\longrightarrow^* Q$).

More precisely, making a lazy use of the non-logical axioms forming the theory, a proof search in first order IMLL can become reasonable (recall that, with no theory, first order

IMLL is in PSPACE [20]). There are already lots of Linear Logic theorem provers available, especially since LL has become quite popular as a specification language [22, 2, 16, 15], and they can be used for a proof search that will use both forward reasoning for the left side of the sequent (copying the π-calculus reductions) but also backward reasoning for the right part (conclusion guided search).

5.2 Phase semantics

The phase semantics is the natural provability semantics of Linear Logic [14]. It has been recently used with the specific aim of proving safety properties of LCC agents [10, 11, 29] and is currently studied as a paradigm for infinite state systems model-checking.

Using the soundness theorem 4.2, it is possible to use the same technique as that used for LCC agents, to prove safety properties by searching for phase counter-models, which amounts to some kind of abstract interpretation. It is also of course possible to use the encoding given in section 3 and to use then the LCC tools mentioned above.

5.3 Program equivalence

The results given in the previous section already give some program equivalence based on LL equivalence, however, one can, along the lines of what Miller already did in [21], notice that we can enrich the syntax of the agents to use the power of LL and thus obtain some finer equivalences.

If one defines:

$$\langle P \rangle = \{Q \mid Q \text{ is a co-agent s.t. } [\![P]\!] \vdash Q\}$$

Allowing co-agents to be more than just translation of agents ($[\![R]\!]$) but also include the $\top$ constant, one can allow the erasing of some part of a process; the $\&$ operator realizes two checks at the same time, while the $\oplus$ only needs one test or another, ...

6 Conclusion and perspectives

One objective of this paper was to understand in more details the old problem of the $\exists$ encoding of the restriction operator, and to find some way around it, if possible without resorting to all the current work around binders ([13, 25, 6]) in order to be able to use the powerful tools of LL.

Combining a formal encoding of the π-calculus into LCC and the author's results about the semantics of the restriction operator of LCC into ILL made possible to discover the two main results of this paper:

First, that the asynchronous π-calculus is easily encoded in a sound and complete way in the standard LCC, the reverse encoding being impossible.

Second, that the first order quantifier $\exists$, though it does not give directly the restriction operator, provides the means to get it through a simple theory based on one main atom of arity one: *loc*.

From these results it is now possible to understand better the very peculiar nature of the first order quantifiers in LL: they bind variables, but also tend to allow some sort of *weakening* ($A(t) \vdash \exists x A(x)$). If we take this unwanted property away, we are able to reconstruct a structure of nested $\exists$ inside the formulae, corresponding to the sharing of names in π-calculus and thus to some notion of space.

It is now very interesting to continue in that way in order to provide a framework to compare all the current attempts at providing *locations* inside the logic, like in the modal LL of [17] or the spatial logic of [6].

The direct use of the embedding outlined in section 5 is currently under study, especially the use of the phase semantics. More or less, as expected, the complexity of the counter models to exhibit seems to depend a lot on the use of the restriction operator. This new topic needs much more work and could lead to some other view at the restriction.

Another direction of work is a better understanding of the links between the binders of Gabbay and Pitts and the simple existential quantifier, which was until now left aside for the reasons explained before. The first distinction is that of the sequent calculus, which comes for free in the case of the $\exists$, compared to the *completeness*, which on the other hand comes for free for the binders. This seems to show a difference related to the use one makes of the logics: for proof search or for describing objects, but a much deeper comparison is required.

Finally, the very active field of the modeling of bio-chemical networks by process calculi (see for instance [26, 9, 7]) also seems to open a whole range of applications of either the encoding into LCC, using then the already existing tools for LCC, or of a direct LL semantics with the creation of specialized tools for checking properties quite different than the usual simulation-based analyses of the π-calculus processes.

7 Acknowledgements

We would like to thank especially Dale Miller for discussions started in July 2000 that were the initial sparkles of this whole work, and François Fages for his constant advice.

References

[1] J. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.

[2] J. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991.

[3] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96, 1992.

[4] E. Best, F. de Boer, and C. Palamidessi. Concurrent constraint programming with information removal. In *Proceedings of Coordination*, LNCS. Springer-Verlag, 1997.

[5] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA, May 1992.

[6] L. Caires and L. Cardelli. A spatial logic for concurrency (part i). In B. Pierce and N. Kobayashi, editors, *TACS'01: Proceedings of the 4th Symposium on Theoretical Aspects of Computer Science*, volume 2215 of *LNCS*, pages 1–37, Sendai, Japan, October 2001. Springer-Verlag.

[7] N. Chabrier and F. Fages. Symbolic model cheking of biochemical networks. In *CMSB'03: Proceedings of the 1st Workshop on Computational Methods in Systems Biology*, March 2003. to appear.

[8] F. de Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. *ACM-TOPLAS*, 19(5):685–725, 1997.

[9] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, and K. Sonmez. Pathway logic: Symbolic analysis of biological signaling. In *Proceedings of the sixth Pacific Symposium on Biocomputing*, pages 400–412, January 2002.

[10] F. Fages, P. Ruet, and S. Soliman. Phase semantics and verification of concurrent constraint programs. In *Proc. 13th Annual IEEE Symposium on Logic in Computer Science, Indianapolis*, 1998.

[11] F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: operational and phase semantics. *Information and Computation*, 164:14–41, 2001.

[12] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *POPL'96: Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385. ACM Press, 1996.

[13] M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In *LICS'99: Proceedings of the 14th Annual Symposium on Logic In Computer Science*, pages 214–224, Washington, 1999. IEEE Computer Society Press.

[14] J. Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987.

[15] J. Harland, D. Pym, and M. Winikoff. Programming in lygon: An overview. In *Proceedings of the Fifth International Conference on Algebraic Methodology and Software Technology, Munich*, pages 391–405, July 1996.

[16] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[17] N. Kobayashi, T. Shimizu, and A. Yonezawa. Distributed concurrent linear logic programming. *Theoretical Computer Science*, 227:185–220, 1999.

[18] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. In R. H. H. Jr. and T. Ito, editors, *Proceedings of Parallel Symbolic Computing*, volume 748 of *LNCS*, pages 331–336. Springer-Verlag, 1992.

[19] C. Laneve and U. Montanari. Mobility in the cc-paradigm. In *Proc. 17th International Symposium on Mathematical Foundations of Computer Science*, volume 629 of *LNCS*, pages 336–345. Springer-Verlag, 1992.

[20] P. Lincoln and N. Shankar. Proof search in first-order linear logic and other cut-free sequent calculi. In *Proc. 9th Annual IEEE Symposium on Logic in Computer Science, Paris*, 1994.

[21] D. Miller. The pi-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of the 1992 Workshop on Extensions to Logic Programming*, volume 660 of *LNCS*, pages 242–265. Springer-Verlag, 1992.

[22] D. Miller. A multiple-conclusion meta-logic. *A Multiple-Conclusion Meta-Logic*, 165(1):201–232, 1996.

[23] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1), 1992.

[24] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *POPL'97: Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, pages 256–265. ACM Press, 1997.

[25] A. Pitts. Nominal logic: A first order theory of names and binding. In B. Pierce and N. Kobayashi, editors, *TACS'01: Proceedings of the 4th Symposium on Theoretical Aspects of Computer Science*, volume 2215 of *LNCS*, pages 219–235, Sendai, Japan, October 2001. Springer-Verlag.

[26] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proceedings of the fifth Pacific Symposium of Biocomputing*, pages 459–470, 2001.

[27] V. Saraswat. *Concurrent constraint programming.* ACM Doctoral Dissertation Awards. MIT Press, 1993.

[28] V. Saraswat and P. Lincoln. Higher-order linear concurrent constraint programming. Technical report, Xerox Parc, 1992.

[29] S. Soliman. Phase model checking for some linear logic calculi. In H. Nivelle and S. Schultz, editors, *Proceedings of the Second International Workshop of the Implementation of Logics, Havana, Cuba*, MPI-I-2001-2-006, pages 60–80. Max-Planck-Institut für Informatik, December 2001.

[30] S. Soliman. *Programmation concurrente avec contraintes et logique linéaire.* PhD thesis, Université Paris 7, Denis Diderot, 2001.

[31] C. Tse. The design and implementation of an actor language based on linear logic. Master's thesis, MIT, 1994.

[32] B. Victor and J. Parrow. Concurrent constraints in the fusion calculus. In S. Skyum and G. Winskel, editors, *ICALP'98: Proceedings of the 25th International Colloquium on Automata, Languages, and Programming*, volume 1443 of *LNCS*, pages 455–469. Springer-Verlag, July 1998.

A Intuitionistic Linear Logic

We give here a brief description of the intuitionistic version of Linear Logic (ILL) with the full sequent calculus (see [14] for more details).

Definition A.1 (Formulae) *The* intuitionistic formulae *are built from atoms p, q, ... with the multiplicative connectives $\otimes$ (tensor) and $\multimap$ (linear implication), the additive connectives $\&$ (with) and $\oplus$ (plus) the exponential connective $!$ (bang), and the universal $\forall$ and existential $\exists$ quantifiers.*

Definition A.2 (Sequents) *The* intuitionistic sequents *are of the form $\Gamma \vdash A$, where A is a formula and Γ is a multi-set of formulae.*

The sequent calculus is given by the following rules, where the basic idea is that the disappearance of the weakening rule makes the conjunction $\otimes$ count the occurrences of formulae, and the implication $\multimap$ consume its premise:

Axiom - Cut

$$A \vdash A \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Delta, \Gamma \vdash B}$$

Constants

$$\frac{\Gamma \vdash A}{\Gamma, \mathbf{1} \vdash A} \quad \vdash \mathbf{1} \quad \Gamma \vdash \top$$

$$\bot \vdash \quad \frac{\Gamma \vdash}{\Gamma \vdash \bot} \quad \Gamma, \mathbf{0} \vdash A$$

Multiplicatives

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Delta, \Gamma, A \multimap B \vdash C}$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

Additives

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} \qquad \frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C}$$

$$\frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

Bang

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}$$

$$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}$$

Quantifiers

$$\frac{\Gamma, A[t/x] \vdash B}{\Gamma, \forall x A \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\ x \notin fv(\Gamma)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, \exists x A \vdash B}\ x \notin fv(\Gamma, B) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399